IRIS EXCHANGE

IREX III

CONCEPT, EVALUATION PLAN AND API

VERSION 0.2

Patrick Grother

Image Group
Information Access Division
Information Technology Laboratory
National Institute of Standards and Technology

December 21, 2010

Status of this Document

This document is a draft. The entire content is open for public comment.
Comments and questions should be submitted to irex@nist.gov.

Timeline of the IREX III Evaluation

Table 1 - Dates and milestones

August 4, 2011	IREX IV – Provisional commencement of NIST's IREX evaluation. This activity will include 1:1 matching. It may extend to 1:N and other areas. The API in this document includes 1:1 functions.
August 4, 2011	Decision on whether to conduct a further round of IREX III. Announce timeline for that.
July 1, 2011	NIST releases first public report. This report will attribute biometric error rate and processing speed estimates to the names of IREX III participants.
May 28, 2011	Window for 1:N participation closes. Anything received after this deadline will be ignored.
March 31, 2011	Hard deadline for a participants first submission of a 1:N algorithms. If a 1:N algorithm is not received by this date, the provider is excluded from all 1:N participation. This milestone, in the middle of the participation window, is intended to ensure that participants do not wait until the last minute to submit.
February 1, 2011	Window for 1:N participation opens. Shaded green below.
January 21, 2011	NIST releases final API
January 4, 2011	Comments due on revised API
January 6, 2011	NIST releases revised API
January 4, 2011	**Comments due on initial API**
December 16, 2010	NIST releases initial API
November 20, 2010	NIST announces IREX IV

January 2011

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

February 2011

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28					

March 2011

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	**31**		

April 2011

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

May 2011

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

June 2011

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

July 2011

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

August 2011

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

September 2011

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

October 2011

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Table of Contents

List of Figures

List of Tables

Terms and definitions

The abbreviations and acronyms of Table 2 are used in many parts of this document.

Table 2 – Abbreviations

FNIR	False negative identification rate
FPIR	False positive identification rate
FMR	False match rate
FNMR	False non-match rate
Reliability	Measure of how many searches for which an enrolled mate exists are successful.
Selectivity	Measure of how many non-matches are returned in a search when in fact no mate is enrolled.
DET	Detection error tradeoff characteristic: For verification this is a plot of FNMR vs. FMR (sometimes as normal deviates, sometimes on log-scales). For identification this is a plot of FNIR vs. FPIR.
INCITS	InterNational Committee on Information Technology Standards
ISO/IEC 19794	ISO/IEC 19794-6: Information technology — Biometric data interchange formats — Part 6: Iris image data. First edition: 2005-06-15. (See Bibliography entry). Second edition: expected mid 2011, replacing 2005.
I379	INCITS 379:2004 - U.S. precursor to the 19794-5:2005 international standard. Now defunct.
ANSI/NIST Type 17	The most common container for iris images in the law enforcement world.
IREX III	NIST's IRIS EXCHANGE program supporting standards-based iris biometrics
NIST	National Institute of Standards and Technology
PIV	Personal Identity Verification
SC 37	Subcommittee 37 of Joint Technical Committee 1 – developer of biometric standards
SDK	The term Software Development Kit refers to any library software submitted to NIST. This is used synonymously with the terms "implementation" and "implementation under test".

IREX III

1.1. Scope

This document establishes a concept of operations and an application programming interface (API) for evaluation of iris identification implementations submitted to NIST's IREX III evaluation. This document covers only the recognition of two-dimensional still infrared images. See http://iris.nist.gov/irex for all IREX documentation.

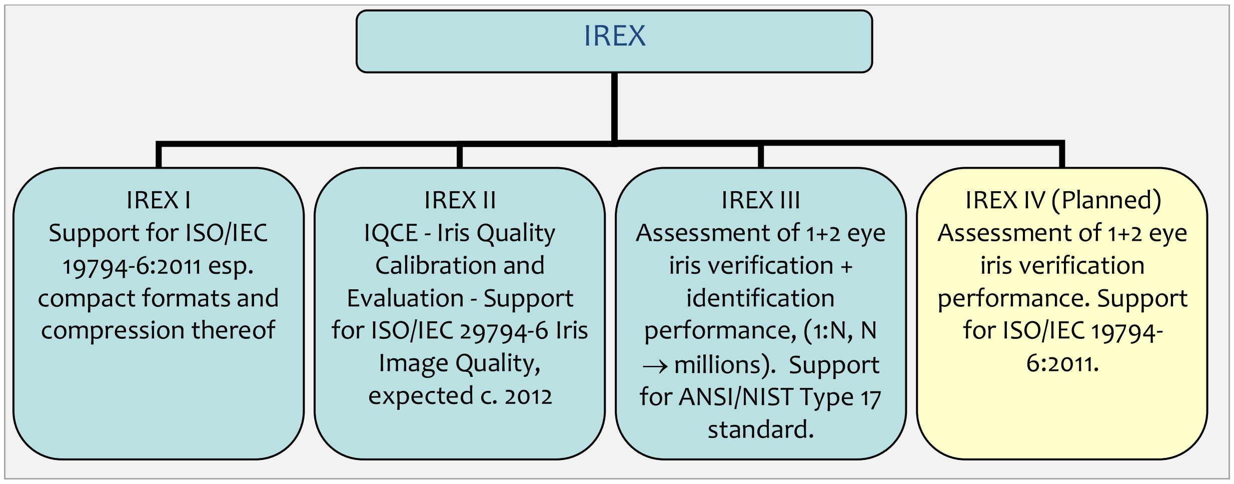
IREX
IREX I Support for ISO/IEC 19794-6:2011 esp. compact formats and compression thereof
IREX II IQCE - Iris Quality Calibration and Evaluation - Support for ISO/IEC 29794-6 Iris Image Quality, expected c. 2012
IREX III Assessment of 1+2 eye iris verification + identification performance, (1:N, N → millions). Support for ANSI/NIST Type 17 standard.
IREX IV (Planned) Assessment of 1+2 eye iris verification performance. Support for ISO/IEC 19794-6:2011.

Figure 1 – Current extent of the IREX Program

1.2. Audience

Universities and commercial entities with a ability to implement a large scale one-to-many iris identification algorithm are invited to participate in the IREX III still-iris test. Organizations with only a one-to-one interest or capability should wait for the IREX IV activity.

Organizations will need to implement the API defined in this document. Participation is open worldwide. There is no charge for participation. While NIST intends to evaluate technologies that could be readily made operational, the test is also open to experimental, prototype and other technologies.

1.3. Purpose and market drivers

This test is intended to support a plural marketplace of iris recognition systems. More specifically, the test is intended to assess 1:N identification performance in as large a population as possible, thereby testing the long-posited promise of iris as very powerful biometric.

While the largest applications, in terms of revenue, have been for one-to-many search for border control and war zone identity management, the use for planetary-scale de-duplication (likely with fingerprints) in the Indian UID program is rapidly growing now.

The test is planned against the backdrop of an expanding marketplace of iris cameras designed to operate in a variety of applications beyond just 1:N de-duplication of an enrollment database. For example:

— some standoff-capture cameras can rapidly image and verify (in a one-to-many mode) high volumes of people;

— some mobile cameras can be preloaded with templates and firmware code for rapid 1:N watchlist.

These applications are differentiated by the population size and other variables.

1.4. Offline testing

While this set of tests is intended as much as possible to mimic operational reality, this remains an offline test executed on databases of images. The intent is to assess the core algorithmic capability of iris recognition algorithms. This test will be conducted purely offline - it does not include a live human-presents-to-camera component. Offline testing is attractive because it allows uniform, fair, repeatable, and efficient evaluation of the underlying technologies. Testing of implementations under a fixed API allows for a detailed set of performance related parameters to be measured. Human-in-the-loop testing is necessary for operational realization of application requirements.

1.5. Phased testing

To support research and development efforts, this testing activity will embed multiple rounds of testing. These test rounds are intended to support improved performance. Once the test commences, NIST will test implementations on a first-come-first-served basis and will return results to providers as expeditiously as possible. Providers may submit revised implementations to NIST only after NIST provides results for the prior implementation. The frequency with which a provider may submit implementations to NIST will depend on the times needed for vendor preparation, transmission to NIST, validation, execution and scoring at NIST, and vendor review and decision processes.

For the number of implementations that may be submitted to NIST see section 1.9.

1.6. Interim reports

The performance of each implementation will be reported in a "score-card". This will be provided to the participant. While the score cards may be used by the provider for arbitrary purposes, they are intended to allow development. The score cards will

– be machine generated (i.e. scripted),
– be provided to participants with identification of their implementation,
– include timing, accuracy and other performance results,
– include results from other participants implementations, but will not identify the other providers,
– be expanded and modified as revised implementations are tested, and as analyses are implemented,
– be generated and released asynchronously with implementation submissions,
– be produced independently of the other status of other providers' implementations,
– be regenerated on-the-fly, primarily whenever any implementation completes testing, or when new analysis is added.

NIST does not intend to release these test reports publicly. NIST may release such information to the U.S. Government test sponsors. While these reports are not intended to be made public, NIST can only request that sponsoring agencies not release this content.

1.7. Final reports

Once NIST terminates the testing rounds, one or more final public reports will be released. NIST may publish

– Reports (typically as numbered NIST Interagency Reports),
– Publications in the academic literature,
– Presentations (typically PowerPoint).

Our intention is that the final test reports will publish results for the all implementations from each participant. The intention is to report results for the most capable implementations (see section 1.12, on metrics). Other results may be included (e.g. in appendices) to show, for example, examples of progress or tradeoffs. **IMPORTANT: Results will be attributed to the providers.**

1.8. Application scenarios

The test will evaluate one-to-many identification implementations[1]. Later it will evaluate 1:1 accuracy also. As described in Table 3, the test is intended to represent close-to-operational use of iris recognition technologies in identification applications in which the enrolled dataset could contain images from up to ten million persons.

Table 3 – Subtests supported under the IREX III still-iris activity

#		A	B
1.	Aspect	1:N identification	Reverse 1:N identification
2.	Enrollment dataset	N enrolled subjects, enrollment images are of good general to good quality.	N enrolled subjects, enrollment images are challenging, possibly acquired under adverse circumstances and with defects.
3.	Identification metrics	Threshold based.	Threshold and rank based.
4.	Prior NIST test references	See IREX at http://iris.nist.gov/irex See MBE face recognition for metrics, analyses at http://face.nist.gov/mbe	
5.	Example application	Open-set identification of an image against a central database, e.g. a search of a mugshot against a database of known criminals.	Forensic identification, a "latent" iris image collected without normal controls, illumination.
6.	Score or feature space normalization support	Any score or feature-based statistical normalization techniques are applied to the enrollment database	Any score or feature-based statistical normalization techniques are applied to the enrollment database
7.	Intended number of subjects	Up to $O(10^7)$ but dependence on N will be computed. From $O(10^2)$ upwards.	$O(10^4)$

1.9. Notes on images

— Images are likely to have dimensions of 640x480 pixels.

— Images will all be collected in the near infra-red.

— Images from more than one sensor will be included.

— Some persons will have images from more than one sensor.

— Some images are of poor quality. NIST will target the natural population. NIST will secondarily attempt to control for non-ideal variations.

— Some images were collected outdoors. Pupil radius may be small.

EDITOR'S NOTE :: More information in second edition of this document.

1.10. Options for participation

The following rules apply:

— A participant must properly follow, complete and submit the Annex A Participation Agreement. This must be done once. It is not necessary to do this for each submitted implementation.

— All participants shall submit at least one class A implementation before the midpoint date given in Table 1.

— Submission of class B implementation is optional.

— All participants shall submit at least one class A implementations labeled slow. Operationally "slow" implementations would be intended to run on blade-class computers (contemporary 64 bit high power CPUs). The implementations are targeted for applications such as 1:N de-duplication transactions.

[1] NIST has previously only modeled identification scenarios. The simplest simulation mimics a 1:N search by conducting N 1:1 comparisons.

- All participants shall submit at least one class A implementations labeled fast. Operationally "fast" implementations would be intended to run on embedded processors (low power consumption, hand held devices). These implementations are targeted for applications such as 1:N (N < 100,000, say) .
- Any implementation shall implement exactly one of the functionalities defined in clause 3.
- At any point in time, the maximum number of implementations undergoing testing at NIST will be two. This is the total of class A plus B. NIST will invite submission of revised implementations when testing of each prior implementation has been completed.
- A provider of an implementation may ask NIST not to repeat feature extraction and enrollment processes. This may expedite testing of an implementation because NIST can proceed directly to identification and verification trials. NIST cannot conduct surveys over runtime parameters - NIST must limit the extent to which participants are able to train on the test data.

1.11. Use of multiple images per person

Some tests will proceed with

- K = 1 image per person. This could be labeled as a left eye (L), a right eye (R), or unknown (U).
- K = 2 image per person. These might be labeled as any combination of L, R, and U.
- K = a random number of images per person. These might be labeled as any combination of L, R and U.

1.12. Core accuracy metrics

For identification testing, the test will target open-universe applications such as de-duplication and watch-lists. It will not address the closed-set task because it is operationally uncommon.

While some one-to-many applications operate on the assumption that a candidate list of identities will be reviewed by a human examiner, for which rank-based metrics are relevant, this test will primarily target *lights-out* identification i.e. the iris identification system operates on its own, making decisions against some threshold. NIST will typically apply score-based identification metrics as defined in the first two rows of Table 4. The analysis will survey over threshold values. The analysis may also use rank, as in the last two rows of the Table. Plots of the two error rates, parametric on threshold, will be the primary reporting mechanism.

Table 4 – Summary of accuracy metrics

	Application	Metric		
A	1:N Identification Primary identification metric.	FPIR	=	Fraction of searches that do not have an enrolled mate for which one or more candidate list entries exceed a threshold
		FNIR	=	Fraction of searches that have an enrolled mate for which the mate is below a threshold
B	1:N Identification (with rank criteria) Secondary identification metric	FPIR	=	Fraction of searches that do not have an enrolled mate for which one or more candidate list entries exceed a threshold
		FNIR	=	Fraction of searches that have an enrolled mate for which the mate is not in the best R ranks *and* at or above a threshold

NOTE: The metric on line B is a special case of the metric on line C: the rank condition is relaxed ($R \rightarrow N$). Metric B is the primary metric of interest because the target application does not include a rank criterion.

FPIR and FMR will usually be estimated using probe images for which there is no enrolled mate.

NIST will extend the analysis in other areas, with other metrics, and in response to the experimental data and results.

1.13. Quality based exclusion

NIST will examine the effectiveness of iris image quality scores. These are computed from input images during feature extraction. The planned analyses relate to accuracy prediction:

- The default method will be the error vs. reject analysis document in P. Grother and E. Tabassi, *Performance of biometric quality measures*, IEEE Trans. PAMI, 29:531–543, 2007.
- NIST will survey over an additional parameter, β, the fraction of images excluded from a subsequent computation of the DET characteristic. The images excluded will be those with the lowest scalar quality value reported by the implementation during template generation. Quality-based exclusion is valuable in multimodal applications, where an alternative biometric can be used when an iris is automatically judged to be poor.

The primary target application will be 1:N de-duplication of a large database.

Analyses other than for the default case may be conducted.

1.14. Reporting of failure to enroll, acquire, process

FTA and FTE have different meanings in offline tests such as IREX III versus online tests such as those conducted in PACS scenario test with humans interacting with biometric readers.

In IREX III, soft failures, where the algorithm elects to not produce a template (e.g. on image quality grounds) shall be treated identically to hard failures, where the algorithm crashes or hangs. In IREX III, any failure to convert images into templates shall be counted as a "failure to enroll" (FTE) and reported as such.

1.15. Matching of empty, broken and missing templates

After a soft failure, the template generator may return an empty (0 byte) template or a short one (a few bytes). In all cases NIST will pass the template to the matching / searching functions which must handle such templates transparently.

After a hard failure, the absence of an output template generator will cause NIST will pass to an empty (0 bytes) template to the matching / searching functions which must handle such templates transparently.

When 1 or more of the templates passed to a one-to-many search are empty or short, the matching / searching functions shall produce a value of -1, and return the appropriate non-zero error code. Such events will be included in the reported performance estimates, as follows:

- In a negative identification system, where a person claims not to be enrolled (e.g. a border crossing watchlist system equipped designed to detect and reject previously deported travelers), the one-to-many search should flag an empty input template (from a traveler wearing contact lenses, for example). NIST will simulate this outcome by setting the score to a low (i.e. genuine) value to force a hit on the database. Such an occurrence would prompt secondary inspection at a border crossing. NOTE: If the image actually had an enrolled mate then this will benefit the accuracy estimate of the implementation under test. If the image did not have an enrolled mate then this will penalize the implementation.
- In a positive access control application e.g. gymnasium access without any claim of identity, a correct one-to-many search should result in a rejection of an empty input template – NIST will simulate this outcome by setting the score to a high value for searches in which a non-zero error is reported.

1.16. Reporting of template size

Because template size is influential on storage requirements and computational efficiency, this API supports measurement of template size. NIST will report statistics on the actual sizes of templates produced by iris recognition implementations submitted to IREX III. Template sizes were reported in [IREX}.

1.17. Reporting of runtime memory usage

NIST will report the amount of memory used during one-to-many searches. That is NIST will not rely on the naïve first order estimate of this, i.e. N times the enrollment template size, plus the size of the search template).

1.18. Reporting of computational efficiency

As with other tests, NIST will compute and report recognition accuracy. In addition, NIST will also report timing statistics for all core functions of the submitted implementation implementations. This includes feature extraction, and 1:1 and 1:N recognition. For an example of how efficiency can be reported, see [IREX].

NIST will plot 1:N search duration as a function of N. Some face identification implementations [MBE] scale as N^b with $b < 1$. Indexing approaches proposed for iris recognition [HAO, UVW] may offer behavior

EDITOR'S QUESTION TO PROVIDERS: Is there any need to support measurement of speed of K > 1 simultaneous, "batched" searches against an N template enrollment. There have been claims that the time taken to execute a 1:N search of 20 templates (e.g. from K = 20 persons) in a single function invocation is less than 20 times the duration of searching 1.

1.19. Exploring the accuracy-speed trade-space

Organizations are encouraged to enter at least two implementations per class. These should be "fast-less-accurate" and "slow-more-accurate" with perhaps a factor of three between the speeds[2]. Speed will be reported alongside some discussion that iris recognition algorithms can run on back-office blade clusters, and on embedded devices such as hand held cameras. Participants are cautioned that NIST will note that algorithms that are slow on blades will be even slower on embedded devices.

While NIST cannot force submission of "fast vs. slow" variants, participants may choose to submit variants on some other axis (e.g. "experimental" vs. "mature").

1.20. Hardware specification

NIST intends to support high performance by specifying the runtime hardware beforehand. NIST will execute the test on high-end PC-class computers. These machines have 4-cpus, each of which has 4 cores. These blades are labeled Dell M905 equipped with 4x Quad Core AMD Opteron 8376HE processors[3] running at 2.3GHz. Each CPU has 512K cache. The bus runs at 667 Mhz. The main memory is 192 GB Memory as 24 8GB modules. Sixteen processes can be run without time slicing.

NIST is requiring use of 64 bit implementations throughout. This will support large memory allocation to support 1:N identification task with image counts in the millions. If all templates were to be held in memory, the 192GB capacity implies a limit of 20KB per template, for a 10 million image enrollment. Note that while the API allows read access of the disk during the 1:N search, the disk is, of course, relatively slow.

Some of the section 3 API calls allow the implementation to write persistent data to hard disk. The amount of data shall not exceed 200 kilobytes per enrolled image.

NIST will respond to prospective participants' questions on the hardware, by amending this section.

1.21. Operating system and compilation environment

All submitted implementations shall run on CentOS 5.5 which runs Linux kernel 2.6.18-194. http://www.centos.org/

NIST will link the provided library file(s) to our ISO 98/99 C/C++ language test drivers. Participants are required to provide their library in a format that is linkable using gcc version 4.1.2[4]. The standard libraries are:

---

[2] See the FNMR vs. time plots in Figure 18 of [IREX].

[3] cat /proc/cpuinfo returns fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm 3wext 3dnow constant_tsc nonstop_tsc pni cx16 popcnt lahf_lm cmp_legacy svm extapic cr8_legacy altmovcr8 abm sse4a misalignsse 3dnowprefetch osvw

[4] The command "gcc –v" gives the following output

```
Using built-in specs.  Target: x86_64-redhat-linux
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --enable-shared --enable-threads=posix --enable-checking=release --with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-libgcj-multifile --enable-languages=c,c++,objc,obj-c++,java,fortran,ada --enable-java-awt=gtk --disable-dssi --enable-plugin --with-java-home=/usr/lib/jvm/java-1.4.2-gcj-1.4.2.0/jre --with-cpu=generic --host=x86_64-redhat-linux
Thread model: posix  gcc version 4.1.2 20080704 (Red Hat 4.1.2-48)
```

```
/usr/lib64/libstdc++.so.6.0.8     lib/libc.so.6 -> libc-2.5.so     /lib/libm.so.6 -> libm-2.5.so
```

A typical link line might be

```
gcc -I. -Wall -m64 -o i3test  i3test.c  -L.  -lirex_Enron_A_07  -lpthread
```

1.22. Threaded computations

Table 5 shows the limits on the numbers of threads an iris recognition implementation may use. In many cases threading is not permitted (i.e. T=1) because NIST will parallelize the test by dividing the workload across many cores and many machines. For the functions where we allow multi-threading, e.g. in the 1:N test, NIST requires the provider to disclose the maximum number of threads to us. If that number is T, NIST will run the largest integer number of processes, P, in parallel such that $TP \leq 16$.

Table 5 – Number of threads allowed for each application

Function	1:N identification
Feature extraction	1
Finalize enrollment (before 1:1 or 1:N)	$1 \leq T \leq 16$
Identification	$1 \leq T \leq 16$

NIST will not run an implementation from participant X and an implementation from participant Y on the same machine at the same time.

For single-threaded libraries, NIST will run up to 16 processes concurrently. NIST's calling applications are single-threaded. Important to compare threaded implementations with non-threaded implementations

1.23. Time limits

The elemental functions of the implementations shall execute under the time constraints of Table 6. These times limits apply to the function call invocations defined in section 3. Assuming the times are random variables, NIST cannot regulate the maximum value, so the time limits are 90-th percentiles. This means that 90% of all operations should take less than the identified duration.

The time limits apply per image. When K images of a person are present (e.g. one image of each eye), the time limits shall be increased by a factor K.

Table 6 – Processing time limits in milliseconds

Function	1:N identification
Feature extraction for enrollment of a 640x480 pixel image	1000 (1 core)
Feature extraction for identification of a 640x480 pixel image	1000 (1 core)
Finalization of a 1 million template enrollment database	7,200,000 (16 cores)

1.24. Ground truth integrity

Some of the test databases will be derived from operational systems. They may contain ground truth errors in which

— a single person is present under two different identifiers, or

— two persons are present under one identifier, or

— in which a iris is not present in the image.

If these errors are detected, they will be removed. NIST will use aberrant scores (high impostor scores, low genuine scores) to detect such errors. This process will be imperfect, and residual errors are likely. For comparative testing, identical datasets will be used and the presence of errors should give an additive increment to all error rates. For very accurate implementations this will dominate the error rate. NIST intends to attach appropriate caveats to the accuracy results. For prediction of operational performance, the presence of errors gives incorrect estimates of performance.

2. Data structures supporting the API

2.1. Overview

This section describes data structures supporting the API of clause 3. In this iris recognition test, an individual is represented by K ≥ 1 two-dimensional iris images each of which is accompanied by a left-right-unknown eye label. The images used in this test may be conformant to the ISO/IEC 19794-6:2011 and ANSI/NIST ITL 1-2011 Standards. Both standards include the kinds of images specified in Table XX – these differ primarily in their geometric specifications. The kind "k7" is specialized in that it includes eyelid and sclera regions that are masked.

2.1.1. Iris image sets

Table 7 – Kind flags indicating standardized properties

Kind string (lowercase k and a digit)	ISO/IEC 19794-6:2011 Presence	Meaning (R is iris radius)	IREX presence
k0	No	Unconstrained – image has no geometric or other requirements associated with standardized image	IREX III
k1	Yes	Image has iris margin requirements greater than or equal to (0.2R, 0.6R) in y and x directions.	
k2	Yes	As for k1 but images are 640 x 480	IREX III
k3	Yes	Images are centered and have strict margin requirements identical to (0.2R, 0.6R).	IREX IV
k7	Yes	As for k3 but the eyelid and sclera are painted over with a fixed pixel value, prior to lossless or lossy compression.	IREX IV

Table 8 – Structure for a single iris, with metadata

	"C" code fragment	Remarks
1.	`typedef struct siris`	
2.	`{`	
3.	`uint16_t image_width;`	Number of pixels horizontally
4.	`uint16_t image_height;`	Number of pixels vertically
5.	`uint8_t eye;`	Eye labels for the ISO standard. SUBJECT_EYE_UNDEF = 0 (00Hex), SUBJECT_EYE_RIGHT = 1 (01Hex) and SUBJECT_EYE_LEFT = 2 (02Hex)
6.	`uint8_t *data;`	Pointer to raster scanned intensity data, 8 bits per pixel.
7.	`char *kind;`	Properties of the image per Table 7
8.	`} ONEIRIS;`	

Table 9 – Structure for a set of images from a single person

	"C" code fragment	Remarks
1.	`typedef struct miris`	
2.	`{`	
3.	`unsigned int numirises;`	The number of accessible pointers, F, such that the last element is F-1
4.	`ONEIRIS **irises;`	Pointers to F pre-allocated iris images of the same person.
5.	`} MULTIIRIS;`	

2.1.2. Datatype for result of an template generation

When an input iris is converted into a template, the implementation shall populate a data structure identical to that in Table 10. To support two eye usage, the linear array of Table 11 shall be used.

Table 10 – Data structure for the result of a template generation function

	"C" code fragment	Remarks

1.	`typedef struct osiris`	
2.	`{`	
3.	`double iris_radius;`	Estimate of iris radius, in pixels
4.	`uint16_t iris_center_x;`	Estimate of the horizontal coordinate of the iris center
5.	`uint16_t iris_center_y;`	Estimate of the vertical coordinate of the iris center
6.	`double pupil_radius;`	Estimate of pupil radius, in pixels
7.	`uint16_t pupil_center_x;`	Estimate of the horizontal coordinate of the pupil center
8.	`uint16_t pupil_center_y;`	Estimate of the vertical coordinate of the pupil center
9.	`uint8_t quality;`	An assessment of image quality. The legal values are: **[0,100]** - The value should have a monotonic decreasing relationship with false non-match rate anticipated for this sample if it was compared with a pristine image of the same person. So, a low value indicates high expected FNMR. **254** - This value indicates the value was not assigned. **255** - This value indicates a failed attempt to calculate a quality score.
10.	`} ONESEGMENTATION;`	

Table 11 – Structure for a set of images from a single person

	"C" code fragment	Remarks
1.	`typedef struct omiris`	
2.	`{`	
3.	`unsigned int numirises;`	The number of accessible pointers, F, such that the last element is F-1
4.	`ONESEGMENTATION **irises;`	Pointers to F pre-allocated iris images of the same person.
5.	`} MULTISEGMENTATION;`	

2.1.3. Data type for distance scores

Identification and verification functions shall return a measure of the distance between the irises data contained in the two templates. The datatype shall be an eight byte double precision real. The legal range is [0, DBL_MAX], where the DBL_MAX constant is larger than practically needed and defined in the <limits.h> include file. Smaller values indicate more likelihood that the two samples are from the same person.

Providers are cautioned that algorithms that natively produce few unique values (e.g. integers on [0,127]) will be strongly disadvantaged by the inability to set a threshold precisely, as might be required to attain a false match rate of exactly 0.0001, for example.

2.2. File structures for enrolled template collection

An implementation converts iris images into a template, using, for example the "convert_images_to_enrollment_template" function of section 3.4.3. To support the one-to-many identification NIST will concatenate enrollment templates into a single large file. This file is called the EDB (for enrollment database). The EDB is a simple binary concatenation of proprietary templates. There is no header. There are no delimiters. The EDB may extend to hundreds of gigabytes in length

This file will be accompanied by a manifest; this is an ASCII text file documenting the contents of the EDB. The manifest has the format shown as an example in Table 12. If the EDB contains N templates, the manifest will contain N lines. The fields are space (ASCII decimal 32) delimited. There are three fields, all containing numeric integers. Strictly speaking, the third column is redundant.

Table 12 – Enrollment dataset template manifest

Field name	Template ID	Template Length	Position of first byte in EDB
Datatype required	Unsigned decimal integer, not necessarily consecutive, nor starting at 0, nor in any particular order.	Unsigned decimal integer	Unsigned decimal integer
Datatype length required	4 bytes	4 bytes	8 bytes
Example lines of a	90201744	1024	0

manifest file appear to the right. Lines 1, 2, 3 and N appear.	16323202	1536	1024
	7456433	512	2560
	...		
	183838	1024	307200000

This modification of the API (since version 0.5) avoids file system overhead associated with storing millions of individual files.

2.3. Data structure for result of an identification search

All identification searches shall return a candidate list of a NIST-specified length. The list shall be sorted in ascending order of the distance score – i.e. the most similar matching entries are listed first with lowest rank. The data structure shall be that of Table 13.

Table 13 – Structure for a candidate

	"C" code fragment	Remarks
1.	`typedef struct candidate`	
2.	`{`	
3.	`uint8_t failed;`	If the candidate computation failed, this value is set on [1,255]. If the candidate is valid it should be set to 0.
4.	`uint32_t template_id;`	The Template ID integer from the enrollment database manifest defined in clause 2.2.
5.	`double distance_score;`	Required measure of distance between the identification template and the enrolled candidate. Lower scores mean more likelihood that the samples are of the same person. An algorithm is free to assign any non-negative value to a candidate. The distribution of values will have an impact on the appearance of a plot of false-negative and false-positive identification rates.
6.	`double probability;`	Required estimate of the probability that the biometric data and candidate belong to *different* persons, i.e. the probability that a score this small would be observed given that the pair of images are from different people = P(DISTANCE | IMPOSTOR). This value shall be on [0:1]. This is the integral of the expected impostor distribution from 0 to the distance_score, i.e. the expected false match rate.
7.	`} CANDIDATE;`	

3. API Specification

3.1. Implementation identifiers

All implementations shall support the self-identification function of Table 14. This function is required to support internal NIST book-keeping. The version numbers should be distinct between any versions which offer different algorithmic functionality.

Table 14 – Implementation identifiers

Prototype	int32_t get_pid(	
	char *sdk_identifier,	A participant-assigned ID. This shall be different for each submitted implementation.
	char *email_address);	Output
Description	This function retrieves a point-of-contact email address from the implementation under test.	
Output Parameters	sdk_identifier	Version ID code as hexadecimal integer printed to null terminated ASCII string. NIST will allocate exactly 5 bytes for this. This will be used to identify the implementation in the results reports. This value should be changed every time any implementation is submitted to NIST. The value is vendor assigned - format is not regulated by NIST. EXAMPLE: "011A"
	email_address	Point of contact email address as null terminated ASCII string. NIST will allocate at least 64 bytes for this. The implementation shall not allocate.
Return Value	0	Success
	Other	Vendor-defined failure

3.2. Maximum template size

All implementations shall report the maximum expected template sizes. These values will be used by the NIST test harnesses to pre-allocate template data. The values should apply to a single image. For a `MULTIIRIS` containing K images, NIST will allocate K times the value returned. The function call is given in Table 15.

Table 15 – Implementation template size requirements

Prototype	int32_t get_max_template_sizes(	
	uint32_t *max_enrollment_template_size,	Output
	uint32_t *max_recognition_template_size)	Output
Description	This function retrieves the maximum template size needed by the feature extraction routines.	
Output Parameters	max_enrollment_template_size	The maximum possible size, in bytes, of the memory needed to store feature data from a single enrollment image.
	max_recognition_template_size	The maximum possible size, in bytes, of the memory needed to store feature data from a single verification or identification image.
Return Value	0	Success
	Other	Vendor-defined failure

3.3. API for 1:1 Verification

EDITOR's NOTE – Whether to include 1:1 in IREX III has not been determined. This API is not expected to be used in IREX III. It is present as a placeholder for IREX IV. Reviewers might comment on whether 1:1 should be conducted sooner (in IREX III) rather than later (in IREX IV). One-to-one applications will need to support cropped-and-masked standard images.

3.3.1. Scope

The 1:1 testing will proceed in three phases: preparation of enrolment templates; preparation of verification templates; and matching. These are detailed in Table 16.

Table 16 – Functional summary of the 1:1 application

Phase	#	Name	Description	Performance Metrics to be reported by NIST
Initialization	I1	Initialization	Function to allow implementation to read configuration data, if any.	None
Enrollment	E1	Serial enrolment	Given K ≥ 1 input images of an individual, the implementation will create a proprietary enrollment template. NIST will manage storage of these templates. NIST requires that these operations may be executed in a loop in a single process invocation, or as a sequence of independent process invocations, or a mixture of both.	Statistics of the time needed to produce a template. Statistics of template size. Rate of failure to produce a template and rate of erroneous function.
Verification	V1	Serial verification	Given K ≥ 1 input images of an individual, the implementation will create a proprietary verification template. NIST will manage storage of these templates. NIST requires that these operations may be executed in a loop in a single process invocation, or as a sequence of independent process invocations, or a mixture of both.	Statistics of the time needed to produce a template. Statistics of template size. Rate of failure to produce a template and rate of erroneous function.
Matching (i.e. comparison)	C1	Serial matching	Given one proprietary enrollment template and one proprietary verification template, compare these and produce a distance score. NIST requires that these operations may be executed in a loop in a single process invocation, or as a sequence of independent process invocations, or a mixture of both.	Statistics of the time taken to compare two templates. Accuracy measures, primarily reported as DETs.

3.3.2. Initialization of the implementation

Before any template generation or matching calls are made, the NIST test harness will make a call to the initialization of the function in Table 17.

Table 17 – implementation **initialization**

Prototype	int32_t initialize_verification(	
	const char *configuration_location)	Input
Description	This function initializes the implementation under test. It will be called by the NIST application before any call to the Table 18 functions convert_multiiris_to_enrollment_template or convert_multiiris_to_verification_template. The implementation under test should set all parameters.	
Input Parameters	configuration_location	A **read-only** directory containing any vendor-supplied configuration parameters or run-time data files. The name of this directory is assigned by NIST. It is not hardwired by the provider. The names of the files in this directory are hardwired in the implementation and are unrestricted.
Output Parameters	none	
Return Value	0	Success
	2	Vendor provided configuration files are not readable in the indicated location.

	8	The descriptions are unexpected, or unusable.
	Other	Vendor-defined failure

3.3.3. Template generation

The functions of Table 18 support role-specific generation of a template data. The format of the templates is entirely proprietary.

Table 18 – Template generation

Prototypes	int32_t convert_multiiris_to_enrollment_template(	
	const MULTIIRIS *input_irises,	Input
	MULTISEGMENTATION *output_properties,	Output
	uint32_t *template_size,	Output
	uint8_t *proprietary_template);	Output
	int32_t convert_multiiris_to_verification_template(	
	const MULTIIRIS *input_irises,	Input
	MULTISEGMENTATION *output_properties,	Output
	uint32_t *template_size,	Output
	uint8_t *proprietary_template);	Output
Description	This function takes a MULTIIRIS, and outputs a proprietary template. It also reports segmentation information in the MULTISEGMENTATION structure. The memory for all output data is allocated by the test harness before the call i.e. the implementation shall not allocate memory. In all cases, even when unable to extract features, the output shall be a template record that may be passed to the match_templates function without error. That is, this routine must internally encode "template creation failed" and the matcher must transparently handle this.	
Input Parameters	input_irises	An instance of a Table 9 structure. Implementations must alter their behavior according to the number of images contained in the structure.
Output Parameters	template_size	The size, in bytes, of the output template
	proprietary_template	The output template. The format is entirely unregulated. NIST will allocate a KT byte buffer for this template: The value K is the number of images in the **MULTIIRIS**; the value T is output by the maximum template size functions of Table 15.
	output_properties	For each input image in the **MULTIIRIS** the function shall return the estimated iris and pupil centers, and image qualities. The calling application will pre-allocate the correct number of **ONESEGMENTATION** structures (i.e. one for each image in the **MULTIIRIS**).
Return Value	0	Success
	2	Elective refusal to process this kind of **MULTIIRIS**
	4	Involuntary failure to extract features (e.g. could not find iris in the input-image)
	6	Elective refusal to produce a template (e.g. insufficient pixels across the iris)
	8	Cannot parse input data (i.e. assertion that input record is non-conformant)
	Other	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

3.3.4. Matching

Matching of one enrollment against one verification template shall be implemented by the function of Table 19.

Table 19 – Template matching

Prototype	int32_t match_templates(	
	const uint8_t *verification_template,	Input
	const uint32_t verification_template_size,	Input
	const uint8_t *enrollment_template,	Input
	const uint32_t enrollment_template_size,	Input
	double *distance);	Output
Description	This function compares two opaque proprietary templates and outputs a distance measure which need not	

	satisfy the metric properties. NIST will allocate memory for this parameter before the call. When either or both of the input templates are the result of a failed template generation (see Table 18), the distance score shall be -1 and the function return value shall be 2.	
Input Parameters	verification_template	A template from convert_multiiris_to_verification_template().
	verification_template_size	The size, in bytes, of the input verification template $0 \le N \le 2^{32} - 1$
	enrollment_template	A template from convert_multiiris_to_enrollment_template().
	enrollment_template_size	The size, in bytes, of the input enrollment template $0 \le N \le 2^{32} - 1$
Output Parameters	distance	A distance score resulting from comparison of the templates, on the range [0,DBL_MAX]. See section 2.1.3.
Return Value	0	Success
	2	Either or both of the input templates were result of failed feature extraction
	Other	Vendor-defined failure

3.4. 1:N Identification

3.4.1. Scope

The 1:N application proceeds in two phases, enrollment and identification. The identification phase includes separate pre-search feature extraction stage, and a search stage.

The design reflects the following *testing* objectives for 1:N implementations.

– support distributed enrollment on multiple machines, with multiple processes running in parallel
– allow recovery after a fatal exception, and measure the number of occurrences
– allow NIST to copy enrollment data onto many machines to support parallel testing
– respect the black-box nature of biometric templates
– extend complete freedom to the provider to use arbitrary algorithms
– support measurement of duration of core function calls
– support measurement of template size

Table 20 – Procedural overview of the identification test

Phase	#	Name	Description	Performance Metrics to be reported by NIST
Enrollment	E1	Initialization	Give the implementation advance notice of the number of individuals and images that will be enrolled. Give the implementation the name of a directory where any provider-supplied configuration data will have been placed by NIST. This location will otherwise be empty. The implementation is permitted **read-write-delete access** to the enrollment directory during this phase. The implementation is permitted read-only access to the configuration directory. After enrollment, NIST may rename and relocate the enrollment directory - the implementation should not depend on the name of the enrollment directory.	
	E2	Parallel Enrollment	For each of N individuals, pass multiple images of the individual to the implementation for conversion to a combined template. The implementation will return a template to the calling application. The implementation is permitted **read-only access** to the enrollment directory during this phase. NIST's calling application will be responsible for storing all templates as binary files. These will not be available to the implementation during this enrollment phase. Multiple instances of the calling application may run simultaneously or sequentially. These may be executing on different computers. The same person will not be enrolled twice.	Statistics of the times needed to enroll an individual. Statistics of the sizes of created templates. The incidence of failed template creations.
	E3	Finalization	Permanently finalize the enrollment directory. This supports, for example, adaptation of the image-processing functions, adaptation of the representation, writing of a manifest, indexing, and computation of statistical information over the enrollment dataset. The implementation is permitted **read-write-delete access** to the enrollment directory during this phase.	Size of the enrollment database as a function of population size N and the number of images. Duration of this operation. The time needed to execute this function shall be reported with the preceding enrollment times.

Pre-search	S1	Initialization	Tell the implementation the location of an enrollment directory. The implementation could look at the enrollment data. The implementation is permitted **read-only access** to the enrollment directory during this phase. Statistics of the time needed for this operation.	Statistics of the time needed for this operation.
	S2	Template preparation	For each probe, create a template from a set of input images. This operation will generally be conducted in a separate process invocation to step S2. The implementation is **permitted no access** to the enrollment directory during this phase. The result of this step is a search template.	Statistics of the time needed for this operation. Statistics of the size of the search template.
Search	S3	Initialization	Tell the implementation the location of an enrollment directory. The implementation should read all or some of the enrolled data into main memory, so that searches can commence. The implementation is permitted **read-only access** to the enrollment directory during this phase.	Statistics of the time needed for this operation.
	S4	Search	A template is searched against the enrolment database. The implementation is permitted **read-only access** to the enrollment directory during this phase.	Statistics of the time needed for this operation. Accuracy metrics - Type I + II error rates. Failure rates.

3.4.2. Initialization of the enrollment session

Before any enrollment feature extraction calls are made, the NIST test harness will call the initialization function of Table 21.

Table 21 – Enrollment initialization

Prototype	int32_t initialize_enrollment_session(	
	const char *configuration_location,	Input
	const char *enrollment_directory,	Input
	const uint32_t num_persons,	Input
	const uint32_t num_images)	Input
Description	This function initializes the implementation under test and sets all needed parameters. This function will be called N=1 times by the NIST application immediately before any M ≥ 1 calls to convert_multiiris_to_enrollment_template. The implementation should tolerate execution of P > 1 processes on the same machine each of which may be reading and writing to the enrollment directory. This function may be called P times and these may be running simultaneously and in parallel.	
Input Parameters	configuration_location	A **read-only** directory containing any vendor-supplied configuration parameters or run-time data files.
	enrollment_directory	The directory will be initially empty, but may have been initialized and populated by separate invocations of the enrollment process. When this function is called, the implementation may populate this folder in any manner it sees fit. Permissions will be read-write-delete.
	num_persons	The number of persons who will be enrolled $0 \le N \le 2^{32} - 1$ (e.g. 1million)
	num_images	The total number of images that will be enrolled, summed over all identities $0 \le M \le 2^{32} - 1$ (e.g. 1.8 million)
Output Parameters	none	
Return Value	0	Success
	2	The configuration data is missing, unreadable, or in an unexpected format.
	4	An operation on the enrollment directory failed (e.g. permission, space).

	6	The implementation cannot support the number of persons or images.
	8	The descriptions are unexpected, or unusable.
	Other	Vendor-defined failure

3.4.3. Enrollment

A **MULTIIRIS** is converted to a single enrollment template using the function of Table 22.

Table 22 – Enrollment feature extraction

Prototypes	int32_t convert_multiiris_to_enrollment_template(		
	const MULTIIRIS *input_irises,		Input
	MULTISEGMENTATION *output_properties,		Output
	uint32_t *template_size,		Output
	uint8_t *proprietary_template);		Output
Description	This function takes a **MULTIIRIS**, and outputs a proprietary template. The memory for the output template is allocated by the NIST test harness before the call i.e. the implementation shall not allocate memory for the result. If the function executes correctly (i.e. returns a zero exit status), the NIST calling application will store the template. The NIST application will concatenate the templates and pass the result to the enrollment finalization function (see section 3.4.4). If the function gives a non-zero exit status: – If the exit status is 8, NIST will debug, otherwise – the test driver will ignore the output template (the template may have any size including zero) – the event will be counted as a failure to enroll. Such an event means that this person can never be identified correctly. IMPORTANT. NIST's application writes the template to disk. The implementation must not attempt writes to the enrollment directory (nor to other resources). Any data needed during subsequent searches should be included in the template, or created from the templates during the enrollment finalization function of section 3.4.4.		
Input Parameters	input_irises	An instance of a Table 9 structure. Implementations must alter their behavior according to the number of images contained in the structure.	
Output Parameters	output_properties	For each input image in the **MULTIIRIS** the function shall return the estimated iris and pupil centers, and image qualities. The calling application will pre-allocate the correct number of **ONESEGMENTATION** structures (i.e. one for each image in the **MULTIIRIS**).	
	template_size	The size, in bytes, of the output template	
	proprietary_template	The format is entirely unregulated. NIST will allocate a KT byte buffer for this template: The value K is the number of images in the **MULTIIRIS**; the value T is output by the maximum enrollment template size function of Table 15.	
Return Value	0	Success	
	2	Elective refusal to process this kind of **MULTIIRIS**	
	4	Involuntary failure to extract features (e.g. could not find iris in the input-image)	
	6	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)	
	8	Cannot parse input data (i.e. assertion that input record is non-conformant)	
	Other	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.	

3.4.4. Finalize enrollment

NIST will write an application around the sole function call of Table 23. Implementations shall not require calls to any other (initialization) functions.

After all templates have been created, in prior processes, the function of Table 23 will be called. This freezes the enrollment data. After this call the enrollment dataset will be forever read-only. This API does not support interleaved enrollment and search phases.

The function allows the implementation to conduct, for example, statistical processing of the feature data, indexing and data re-organization. The function may alter the file structure. It may increase or decrease the size of the stored data. No output is expected from this function, except a return code.

Table 23 – Enrollment finalization

Prototypes	int32_t finalize_enrollment (	
	const char *enrolment directory,	Input
	const char *edb_name,	Input
	const char *edb_manifest_name);	Input
Description	This function takes the name of the top-level directory where enrollment database (EDB) and its manifest have been stored. These are described in section 2.2. The enrollment directory permissions will be read + write. The function supports post-enrollment vendor-optional book-keeping operations and statistical processing. The function will generally be called in a separate process after all the enrollment processes are complete. This function should be tolerant of being called two or more times. Second and third invocations should probably do nothing.	
Input Parameters	enrollment_directory	The top-level directory in which enrollment data was placed. This variable allows an implementation to locate any private initialization data it elected to place in the directory.
	edb_name	The name of a single file containing concatenated templates, i.e. the EDB of section 2.2. While the file will have read-write-delete permission, the implementation should only alter the file if it preserves the necessary content, in other files for example. The file may be opened directly. It is not necessary to prepend a directory name.
	edb_manifest_name	The name of a single file containing the EDB manifest of section 2.2. The file may be opened directly. It is not necessary to prepend a directory name.
Output Parameters	None	
Return Value	0	Success
	2	Cannot locate the input data - the input files or names seem incorrect.
	4	An operation on the enrollment directory failed (e.g. permission, space).
	6	One or more template files are in an incorrect format.
	Other	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

3.4.5. Pre-search feature extraction

3.4.5.1. Scope

This section defines the API for production of templates from search images. Templates produced during enrollment will not be used during search. This allows role-specific asymmetric templates. NIST will write an application around function calls of 3.4.5.2 and 3.4.5.3. Implementations shall not require calls to any other (initialization) functions.

3.4.5.2. Initialization

Before a **MULTIIRIS** is sent to the identification feature extraction function, the test harness will call the initialization function in Table 24.

Table 24 – Identification feature extraction initialization

Prototype	int32_t initialize_feature_extraction_session(	
	const char * configuration_location,	Input
	const char * enrolment directory);	Input
Description	This function initializes the implementation under test and sets all needed parameters. This function will be called N=1 times by the NIST application immediately before any M ≥ 1 calls to convert_multiiris_to_identification_template. The implementation should tolerate execution of P > 1 processes on the same machine each of which can read the configuration directory. This function may be called P times and these may be running simultaneously and in parallel. The implementation has read-only access to its prior enrollment data.	
Input Parameters	configuration_location	A **read-only** directory containing any vendor-supplied configuration parameters or run-time data files.
	enrollment_directory	The top-level directory in which enrollment data was placed and then finalized by the implementation. The implementation can parameterize subsequent template production on the basis of the enrolled dataset.
Output Parameters	none	
Return Value	0	Success
	2	The configuration data is missing, unreadable, or in an unexpected format.
	4	An operation on the enrollment directory failed (e.g. permission).
	Other	Vendor-defined failure

3.4.5.3. Feature extraction

A **MULTIIRIS** is converted to an atomic identification template using the function of Table 25. The result may be stored by NIST, or used immediately. The implementation shall not attempt to store any data.

Table 25 – Identification feature extraction

Prototypes	int32_t convert_multiiris_to_identification_template(	
	const MULTIIRIS *input_irises,	Input
	MULTISEGMENTATION *output_properties,	Output
	uint32_t *template_size,	Output
	uint8_t *identification_template);	Output
Description	This function takes a **MULTIIRIS**, and outputs a proprietary template. The memory for the output template is allocated by the NIST test harness before the call i.e. the implementation shall not allocate memory for the result. If the function executes correctly, it returns a zero exit status. The NIST calling application may commit the template to permanent storage, or may keep it only in memory (the vendor implementation does not need to know). If the function returns a non-zero exit status, the output template will be not be used in subsequent search operations. The function shall not have access to the enrollment data, nor shall it attempt access.	
Input Parameters	input_irises	An instance of a Table 9 structure. Implementations must alter their behavior according to the number of images contained in the structure.

Output Parameters	output_properties	For each input image in the **`MULTIIRIS`** the function shall return the estimated iris and pupil centers, and image qualities. The calling application will pre-allocate the correct number of **`ONESEGMENTATION`** structures (i.e. one for each image in the **`MULTIIRIS`**).
	template_size	The size, in bytes, of the output template
	identification_template	The output template for a subsequent identification search. The format is entirely unregulated. NIST will allocate a KT byte buffer for this template: The value K is the number of images in the input **`MULTIIRIS`**; the value T is output by the maximum enrollment template size function of Table 15.
Return Value	0	Success
	2	Elective refusal to process this kind of **`MULTIIRIS`**
	4	Involuntary failure to extract features (e.g. could not find iris in the input-image)
	6	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
	8	Cannot parse input data (i.e. assertion that input record is non-conformant)
	Other	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

3.4.6. Search

3.4.6.1. Scope

Once search templates have been produced, they may be searched against an enrollment database. NIST will write an application around function calls of 3.4.6.2 and 3.4.6.3. Implementations shall not require calls to any other (initialization) functions.

3.4.6.2. Initialization

The function of Table 26 will be called once prior to one or more calls of the searching function of Table 27. The function might set static internal variables so that the enrollment database is available to the subsequent identification searches.

Table 26 – Identification initialization

Prototype	int32_t initialize_identification_session(	
	const char *configuration_location,	Input
	const char *enrollment_directory);	Input
Description	This function reads whatever content is present in the enrollment_directory, for example a manifest placed there by the finalize_enrollment function.	
Input Parameters	configuration_location	A read-only directory containing any vendor-supplied configuration parameters or run-time data files.
	enrollment_directory	The top-level directory in which enrollment data was placed.
Return Value	0	Success
	Other	Vendor-defined failure

3.4.6.3. Search

The function of Table 27 compares a proprietary identification template against the enrollment data and returns a candidate list.

Table 27 – Identification search

Prototype	int32_t identify_template(	
	const uint8_t *identification_template,	Input
	const uint32_t identification_template_size,	Input
	const uint32_t candidate_list_length,	Input
	CANDIDATE * const *candidate_list);	Output
Description	This function searches a template against the enrollment set, and outputs a list of candidates. NIST will allocate memory for the candidates before the call.	
Input Parameters	identification_template	A template from convert_multiiris_to_identification_template() - If the value returned by that function was non-zero the contents of identification_template will not be used and this function (i.e. identify_template) will not be called.
	identification_template_size	The size, in bytes, of the input identification template $0 \le N \le 2^{32} - 1$
	candidate_list_length	The number of candidates the search should return
Output Parameters	candidate_list	An array of "candidate_list_length" pointers to candidates. The datatype is defined in section 2.3. Each candidate shall be populated by the implementation. The candidates shall appear in ascending order of distance score - i.e. most similar entries appear first.
Return Value	0	Success
	2	The input template was defective.
	Other	Vendor-defined failure

NOTE: Ordinarily the calling application will set the input candidate list length to operationally typical values, say $0 \le L \le 100$, and L << N. We may therefore extend the candidate list length such that L approaches N.

3.5. Software and Documentation

3.5.1. Implementation library and platform requirements

Participants shall provide NIST with binary code only (i.e. no source code). Header files (".h") should not be necessary. They are allowed, but these shall not contain intellectual property of the company nor any material that is otherwise proprietary. It is preferred that the implementation be submitted in the form of a single static library. However, dynamic and shared library files are permitted.

The core library shall be named according to Table 28. If necessary additional dynamic or shared library files may be submitted that support this "core" library file (i.e. the "core" library file may have dependencies implemented in these other libraries).

Intel IPP libraries are not permitted, and will not be supplied.

Access to any GPUs is not permitted.

Table 28 – Implementation library filename convention

Form	libMBE_provider_class_sequence.ending				
Underscore delimited parts of the filename	libMBE	provider	classes	sequence	ending
Description	First part of the name, required to be this.	Single word name of the main provider EXAMPLE: Acme	Functional class in Table 3. EXAMPLE: A	A two digit decimal identifier to start at 00 and increment by 1 every time any implementation is sent to NIST. EXAMPLE: 07	Either .so or .a
Example	libIREX_Enron_A_07.a				

NIST will report the size of the supplied libraries.

3.5.2. Configuration and vendor-defined data

The implementation under test may be supplied with configuration files and supporting data files. The total size of the implementation, that is all libraries, include files, data files and initialization files shall be less than or equal to 1 073 741 824 bytes = 1024^3 bytes.

NIST will report the size of the supplied configuration files.

3.5.3. Linking

On request, NIST will allow use of "g++" for linking, but the API must have "C" linkage. The Standard C++ library is available[5]. The prototypes of this document will be written to a file "irex.h" which will be included via

```
extern "C"
{
#include <irex.h>
}
```

NIST will handle all input of images via NETPBM or PNG libraries..

[5] This includes the compiler that installs with RedHat, which is Target: x86_64-redhat-linux configured with: ../configure --prefix=/usr --mandir=/usr/share/man -- infodir=/u sr/share/info --enable-shared --enable-threads=posix --enable- checking=release - -with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-libgcj-multifile --enable-languages=c,c++,objc,obj-c++,java,fortran,ada --enable-java-awt=gtk --disable-dssi --enable-plugin --with-java-home=/usr/lib/jvm/java-1.4.2-gcj-1.4.2.0/jre --with-cpu=generic --host=x86_64-redhat-linux Thread model: posix gcc version 4.1.2 20070626 (Red Hat 4.1.2-14)

The libraries are what shipped with RH 5.1: /usr/lib64/libstdc++.so.6.0.8 lib/libc.so.6 -> libc-2.5.so /lib/libm.so.6 -> libm-2.5.so

All compilation and testing will be performed on x86 platforms. Thus, participants are strongly advised to verify library-level compatibility with gcc (on an equivalent platform) prior to submitting their software to NIST to avoid linkage problems later on (e.g. symbol name and calling convention mismatches, incorrect binary file formats, etc.).

NIST will not allow or support Intel Integrated Performance Primitives (Intel IPP) and "icc" compiled libraries. See the processor specifications in section 1.20.

For this test, Windows machines will not be used. Windows-compiled libraries are not permitted. All software must run under LINUX.

Dependencies on external dynamic/shared libraries such as compiler-specific development environment libraries are discouraged. If absolutely necessary, external libraries must be provided to NIST upon prior approval by the Test Liaison.

3.5.4. Installation and Usage

The implementation must install easily (i.e. one installation step with no participant interaction required) to be tested, and shall be executable on any number of machines without requiring additional machine-specific license control procedures or activation.

The implementation shall be installable using simple file copy methods. It shall not require the use of a separate installation program.

The implementation shall neither implement nor enforce any usage controls or limits based on licenses, number of executions, presence of temporary files, etc. The implementations shall remain operable until April 30 2011.

Hardware (e.g. USB) activation dongles are not acceptable.

3.5.5. Hard disk space

IREX III participants should inform NIST if their implementations require more than 100K of persistent storage, per enrolled image on average.

3.5.6. Documentation

Participants shall provide complete documentation of the implementation and detail any additional functionality or behavior beyond that specified here. The documentation must define all (non-zero) vendor-defined error or warning return codes.

3.5.7. Modes of operation

Individual implementations provided shall not include multiple “modes” of operation, or algorithm variations. No switches or options will be tolerated within one library. For example, the use of two different “coders” by an feature extractor must be split across two separate implementation libraries, and two separate submissions.

3.6. Runtime behavior

3.6.1. Interactive behavior

The implementation will be tested in non-interactive “batch” mode (i.e. without terminal support). Thus, the submitted library shall not use any interactive functions such as graphical user interface (GUI) calls, or any other calls which require terminal interaction e.g. reads from “standard input”.

3.6.2. Error codes and status messages

The implementation will be tested in non-interactive “batch” mod, without terminal support. Thus, the submitted library shall run quietly, i.e. it should not write messages to "standard error" and shall not write to “standard output”. An implementation may write debugging messages to a log file - the name of the file must be declared in documentation.

3.6.3. Exception Handling

The application should include error/exception handling so that in the case of a fatal error, the return code is still provided to the calling application.

3.6.4. External communication

Processes running on NIST hosts shall not side-effect the runtime environment in any manner, except for memory allocation and release. Implementations shall not write any data to external resource (e.g. server, file, connection, or other process). Implementations shall not attempt to read any resource other than those explicitly allowed in this document. If detected, NIST reserves the right to cease evaluation of all implementations from the supplier, notification to the provider, and documentation of the activity in published reports.

3.6.5. Stateful behavior

All components in this test shall be stateless, except as noted. This applies to iris detection, feature extraction and matching. Thus, all functions should give identical output, for a given input, independent of the runtime history. NIST will institute appropriate tests to detect stateful behavior. If detected, NIST reserves the right to cease evaluation all implementations from the supplier, notification to the provider, and documentation of the activity in published reports.

4. References

MBE	P. Grother, G .W. Quinn, and P. J. Phillips, Multiple-Biometric Evaluation (MBE) 2010, *Report on the Evaluation of 2D Still-Image Face Recognition Algorithms*, NIST Interagency Report 7709, Released June 22, 2010. Revised August 23, 2010. http://face.nist.gov/mbe
HAO	F. Hao, J. Daugman, and Z. Piotr. *A fast search algorithm for a large fuzzy database*. IEEE Transactions on Information Forensics and Security, 3(2):203–212, June 2008.
IREX	P. Grother, E. Tabassi, G. W. Quinn, W. Salamon, Iris Exchange I (IREX I), *Performance of Iris Recognition Algorithms on Standard Images*, NIST Interagency Report 7629, October 22, 2009. http://iris.nist.gov/irex
PERFSTD INTEROP	ISO/IEC 19795-4 — Biometric Performance Testing and Reporting — Part 4: Interoperability Performance Testing. Posted as document 37N2370. The standard was published in 2007. It can be purchased from ANSI at http://webstore.ansi.org/.
ISO STD11	ISO/IEC 19794-5:2011 — *Information technology — Biometric data interchange formats — Part 5: Iris image data*. The standard is expected to be completed in January 2011, and formally published in Summer 2011. It will replace the original ISO standard published in 2005. The standard can be purchased from ANSI at http://webstore.ansi.org/
UVW	Rajiv Mukherjee and Arun Ross, *Indexing Iris Images*, in Proc. of International Conference on Pattern Recognition (ICPR), (Tampa, USA), December 2008.

Annex A
Submission of Implementations to IREX III

A.1 Submission of implementations to NIST

NIST requires that all software, data and configuration files submitted by the participants be signed and encrypted. Signing is done with the participant's private key, and encryption is done with the NIST public key. The detailed commands for signing and encrypting are given here: http://iris.nist.gov/irex/crypto_protection.pdf [Link is correct Jan 28 2010].

NIST will validate all submitted materials using the participant's public key, and the authenticity of that key will be verified using the key fingerprint. This fingerprint must be submitted to NIST by writing it on the signed participation agreement.

By encrypting the submissions, we ensure privacy; by signing the submission, we ensure authenticity (the software actually belongs to the submitter). **NIST will not accept into IREX III any submission that is not signed and encrypted. NIST accepts no responsibility for anything that is transmitted to NIST that is not signed and encrypted with the NIST public key.**

A.2 How to participate

Those wishing to participate in IREX III testing must do all of the following, on the schedule listed on Page 2.

— IMPORTANT: Follow the instructions for cryptographic protection of your implementation and data here. http://iris.nist.gov/irex/crypto_protection.pdf

— Send a signed and fully completed copy of the *Application to Participate in the IREX III Evaluation* *(EDITORS NOTE :: To be released January 2011)*. This must identify, and include signatures from, the Responsible Parties as defined in section XX. The properly signed IREX III Application to Participate shall be sent to NIST as a signed then scanned PDF file.

— Provide an implementation (Software Development Kit) library which complies with the API (Application Programmer Interface) specified in this document.

- Encrypted data and implementations below 20MB can be emailed to NIST at irex@nist.gov
- Encrypted data and implementations above 20MB shall be
  - Made available as a file.zip.gpg or file.zip.asc download from a generic webserver[6], or:
  - Mailed as a file.zip.gpg or file.zip.asc on CD / DVD to NIST at this address:

IREX III Test Liaison (A203) 100 Bureau Drive A203/Tech225/Stop 8940 NIST Gaithersburg, MD 20899-8940 USA	In cases where a courier needs a phone number please use NIST shipping and handling on: 301 -- 975 -- 6296.

A.3 Implementation validation

Registered Participants will be provided with a small validation dataset and program available on the website http://iris.nist.gov/irex at XXX (TBA).

The validation test programs shall be compiled by the provider. The output of these programs shall be submitted to NIST.

[6] NIST will not register, or establish any kind of membership, on the provided website.

Prior to submission of the implementation and validation data, the Participant must verify that their software executes on the validation images, and produces correct distance scores and templates.

Software submitted shall implement the IREX III API Specification as detailed in the body of this document.

Upon receipt of the implementation and validation output, NIST will attempt to reproduce the same output by executing the implementation on the validation imagery, using a NIST computer. In the event of disagreement in the output, or other difficulties, the Participant will be notified.